

Programação em Python

JOÃO PAULO MAGALHÃES

JPM@ESTG.IPP.PT

Equipa Docente

Regente: João Paulo Magalhães

Equipa docente:

- Prof. João Paulo Magalhães (jpm@estg.ipp.pt) – LEI e LSIRC

PYTHON



Python

- Multi propósito (Web, GUI, Scripting, etc)
- Orientado a Objetos
- Interpretada
- Focus na facilidade de leitura e produtividade
- Multi plataforma

SINTAXE



Hello World

```
#!/usr/bin/env python  
print "Hello World!"
```

Indentação

- A maior parte das linguagens não quer saber
- A maior parte dos humanos quer saber
 - Tendemos a agrupar coisas similares

Indentação

```
/* Bogus C code */  
if (foo)  
    if (bar)  
        baz(foo, bar);  
else  
    qux();
```

```
/* Bogus C code */  
if (foo)  
if (bar)  
baz(foo, bar);  
else  
qux();
```

O else pertence ao 2º if

Indentação

```
# Python code
if foo:
    if bar:
        baz(foo, bar)
    else:
        qux()
```

O Python adota indentação

Documentar código

```
# uma linha de código comentada (#)
```

```
"""
```

```
Qualquer string não atribuída a uma variável é  
Considerada um comentário.
```

```
Este é um exemplo de um comentário em múltiplas linhas
```

```
"""
```

```
“Esta é uma linha de código comentado”
```

EXECUÇÃO

Execução- Interativa

```
Joas-MBP:tmp jpm$ python
Python 2.7.10 (default, Jul 15 2017, 17:16:57)
[GCC 4.2.1 Compatible Apple LLVM 9.0.0 (clang-900.0.31)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print "Hello World"
Hello World
>>> ^D
Joas-MBP:tmp jpm$
```

Execução - Scripting

```
Joaos-MBP:tmp jpm$ vi ex1.py  
print "Hello World"
```

```
Joaos-MBP:tmp jpm$ python ex1.py  
Hello World  
Joaos-MBP:tmp jpm$
```

TIPOS DE DADOS



Strings

Isto é uma string

nome = "Estamos na \"ESTG\" a ter aulas de ASI"

Isto também é uma string

Nome = 'Estamos na "ESTG" a ter aulas de ASI'

Isto é uma string multi-linha

sites = '''podes-me encontrar online
nos sites Github e twitter. '''

Isto também é uma string multi-linha

bio = """Se não me encontrares online
Podes encontrar-me lá fora. """

Números

Inteiros

ano = 2010

ano = int("2010")

Vírgula flutuante

pi = 3.14159265

pi = float("3.14159265")

Número fixo de casas decimais

from decimal import Decimal

preco = Decimal("0.02")

Nulo

```
dados_opcionais = None
```

Listas

```
# Listas heterogêneas
```

```
favoritos = []
```

```
# Adicionar à lista
```

```
favoritos.append(42)
```

```
# Estendendo a lista com outra lista
```

```
favoritos.extend(["Python", True])
```

```
# É o equivalente a fazer
```

```
favoritos = [42, "Python", True]
```

Listas

```
numeros = [1, 2, 3, 4, 5]
```

```
len(numeros)
```

```
# 5
```

```
numeros[0]
```

```
# 1
```

```
numeros[0:2]
```

```
# [1, 2]
```

```
numeros[2:]
```

```
# [3, 4, 5]
```

Listas- funções

```
a = [1, 2, 3, 4]
```

```
# Funções .reverse() e .sort()
```

```
a.reverse()
```

```
a.sort()
```

```
# Funções .append() e .insert()
```

```
a.append(10)          #[1, 2, 3, 4, 10]
```

```
a.insert(1,20)        #[1, 20, 2, 3, 4]
```

```
# Função .extend()
```

```
a.extend([11,12,13,14])  # [1, 2, 3, 4, 11, 12, 13, 14]
```

Listas - funções

Função .remove()

a = [1, 2, 3, 4, 5, 3]

a.remove(3) # [1, 2, 4, 5, 3] (remove primeiro que encontra)

Função .count()

a = [1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4]

print(a.count(1)) # 3

Função .pop()

a = [1, 1, 3, 4, 5, 3]

a.pop() # [1, 1, 3, 4, 5]

a.pop(2) # [1, 1, 4, 5]

Tuplos (conjunto de dados imutáveis)

```
numeros = (1,2,3,4,5)
```

```
len(numeros)
```

```
# 5
```

```
numeros[0]
```

```
# 1
```

```
numeros[0:2]
```

```
# (1,2)
```

```
numeros[2:]
```

```
# (3, 4, 5)
```

Tuplos (conjunto de dados imutáveis)

```
numeros[0] = 9
```

```
# TypeError: 'tuple' object does not support item assignment
```

Dicionários

```
peessoa = {}
```

```
# Set pela chave / Get pela chave  
peessoa['nome'] = 'Ana Margarida'
```

```
# Update  
peessoa.update({  
    'favoritos': [42, 'comida'],  
    'sexo' : 'masculino',  
})
```

```
# Qualquer objeto inalterável poderá ser usado como chave  
peessoa[42] = 'numero favorito'  
peessoa[(44.47, -73.21)] = 'coordenadas'
```


Dicionários- funções

```
pessoa = {'nome' : 'Joao', 'sexo' : 'masculino'}
```

```
pessoa['nome'] ou pessoa.get('nome')  
# 'Joao'
```

```
pessoa.keys()  
# ['nome', 'sexo']
```

```
pessoa.values()  
# ['masculino', 'Joao']
```

```
pessoa.items()  
# [('sexo', 'masculino'), ('nome', 'Joao')]
```

Dicionários- funções

```
# Função .update()
d = {'a': 1, 'c': 3, 'b': 2}
e = {'p': 10, 'q': 15}
d.update(e)      # {'p': 10, 'a': 1, 'q': 15, 'b': 2, 'c': 3}

# Função .clear()
d = {'a': 1, 'c': 3, 'b': 2}
d.clear()         # {}
```

Booleans

Isto é um booleano

is_python = True

Em python tudo pode ser transformado (cast) num booleano

is_python = bool("qualquer objeto")

Tudo isto é equivalente a Falso

falso = False ou 0 ou "" ou {} ou [] ou None

#Quase todo o resto é equivalente a Verdadeiro

verdadeiro = True ou 1 ou "texto" ou {'a': 'b'} ou ['c', 'd']

OPERADORES



Aritméticos

```
a = 10          # 10
a += 1          # 11
a -= 1          # 10

b = a + 1       # 11
c = a - 1       # 9

d = a * 2       # 20
e = a / 2       # 5
f = a % 3       # 1
g = a ** 2      # 100
```

Manipulação de strings

```
animais = "Gatos " + "Caes "
```

```
animais += "Coelhos"
```

```
# Gatos Caes Coelhos
```

```
fruta = ", ".join(['Maça', 'Banana', 'Laranja'])
```

```
# Maça, Banana, Laranja
```

```
data = ' %s %d %d' % ('Jan', 11, 2018)
```

```
# Jan 11 2018
```

```
nome = '%(nome)s %(sobrenome)s' % { 'nome' : 'Carlos', 'sobrenome' : 'Silva' }
```

```
# Carlos Silva
```

Manipulação de strings- funções

```
# Concatenação com +
```

```
a = "ESTG "
```

```
b = "ASI"
```

```
c = a + b
```

```
# len() e count()
```

```
a = "ESTG"
```

```
print('tamanho da string a = ', len(a))
```

```
print('numero de Ts na string a = ', a.count('T'))
```

Manipulação de strings- funções

```
#Iteração e indexação
```

```
frase = "Nao temos colher"
```

```
for i, c in enumerate(frase)
```

```
    print (i, c)
```

```
# função .strip() e .startswith()
```

```
frase = "    Nao temos colher    "
```

```
n_frase = frase.strip() #Retira os espaços à esquerda e direita
```

```
if (frase.strip().startswith('Na')):
```

```
    print('Começa por Na')
```


Manipulação de strings- funções

```
# funções .upper() e .lower()
frase = "Nao temos colher"
maiusculas = frase.upper()
minusculas = frase.lower()

# função .replace()
palavra = "pois"
n_palavra = palavra.replace('p', 'd')
```

Manipulação de strings- funções

```
# funções .split() e .join()
```

```
frase = "Nao temos colher"
```

```
s = frase.split('o')          # ['Na', ' tem', 's c', 'lher']
```

```
j = ";".join(s)               # Na; tem;s c;lher
```

```
j = "O".join(s)               # NaO temOs cOlher
```

```
# função .splitlines()
```

```
d = """ Um pequeno texto que até  
ocupa várias linhas
```

```
algumas das linhas estão em branco"""
```

```
print(d.splitlines())          # [' Um pequeno texto que até', 'ocupa várias linhas', '', 'algumas das linhas estão em branco']
```

Comparadores lógicos

AND

a and b

OR

a or b

NOT

not a

COMPOSTO

(a and not (b or c))

Comparações de igualdade

Igualdade

1 is 1 (True)

Desigualdade

1 is not '1' (True)

Exemplo

bool(1) (True)

bool(True) (True)

1 and True (True)

1 is True (False)

Comparações aritméticas

Ordem

$a > b$

$a \geq b$

$a < b$

$a \leq b$

Igualdade e Diferença

$a == b$

$a != b$

CONTROLO DO FLUXO



Condicionais

```
nota = 82
if nota >= 90:
    if nota == 100:
        print "A+"
    else:
        print "A"
elif nota >= 80:
    print "B"
elif nota >= 70:
    print "C"
else:
    print "F"
```

Repetitivas- *for*

```
for x in range(10):  
    print x
```

```
frutas = ['Maça', 'Laranja']
```

```
for fruta in frutas:  
    print fruta
```


Repetitivas – *for* estendido

```
países = {'PT' : 'Portugal', 'ES': 'Espanha'}  
  
for chave, valor in países.items():  
    print '%s: %s' % (chave, valor)
```

Repetitivas – *while*

```
x = 0  
  
while x < 100:  
    print x  
    x += 1
```

IO - FICHEIROS



Ficheiros – Abertura, leitura e fecho

`.read()`, com `open()` e `close()` explícitos

```
fp = open('ficheiro.txt')  
conteudo = fp.read()  
fp.close()
```

`.read()`, dentro do bloco de um comando `with()`

```
with open('ficheiro.txt') as fp:  
    conteudo = fp.read()
```

Ficheiros – Abertura, leitura e fecho

`.readlines()` - (separa as linhas para uma lista)

```
with open('ficheiro.txt') as fp:  
    conteudo = fp.readlines()
```

Numa string, `\n` indica a mudança de linha. (Conta como apenas 1 caractere).

Muitas vezes, é necessário elimina-los. Para isso podemos usar a função `.strip()`

```
with open('ficheiro.txt') as fp:  
    conteudo = fp.readlines()  
  
conteudo = [linha.strip() for linha in conteudo]
```

Ficheiros – Abertura, leitura e fecho

Iteração de ficheiros com **for** - (linha-a-linha)

```
with open('ficheiro.txt') as fp:  
    for linha in fp:  
        linha = linha.strip()  
        print('Linha: ', linha)
```

Ficheiros – Escrita

Função `print()` para ficheiros

```
with open('ficheiro_out.txt', 'w') as fp:  
    print('1, 2, 3, experiência, som, som', file=fp)  
    for i in range(30):  
        print(i, i**0.5, file=fp)
```

```
1, 2, 3, experiência, som, som  
0 0.0  
1 1.0  
2 1.4142135623730951  
3 1.7320508075688772  
4 2.0  
...
```

Ficheiros – Escrita

Função `write()` para ficheiros

```
with open('ficheiro_out.txt', 'w') as fp:  
    fp.write('1, 2, 3, experiência, som, som')
```

FUNÇÕES PRÓPRIAS

Funções

As funções podem ser:

- “livres” (por exemplo a função `len()`) ou
- associadas a objetos (por exemplo, `s.split()` que está associada à *string* `s`).

Funções - Exemplo

```
def minha_funcao()  
    """codigo comentado """  
    print ("Hello World")
```

Funções- Argumentos

```
#Posicionais
def add(x,y):
    return x+y

#Keyword
def shout(frase="Yuppiiii!"):
    print (frase)

#Posicionais + Keyword
def echo(texto, prefixo=''):
    print ("%s%s" & (prefixo, texto))
```

Funções – Argumentos arbitrários

```
def nome_funcao(*args, **kwargs):  
    for arg in args:  
        print (arg)  
  
    for key,value in kwargs.items():  
        print (key)  
  
nome_funcao(1, 2, 3, nome="numeros")
```

Funções – Variáveis locais

```
def recta(m, b, x):  
    r1, r0 = m*x, b  
    return r1 + r0  
  
m, b, x = 2.0, 3.0, 2.0  
res = recta(m, b, x)  
  
print('Para x =', x, 'm =', m, 'b =',  
b)  
print('m*x =', r1, 'b =', r0)  
print('Resultado:', res)
```

MÓDULOS



Módulos

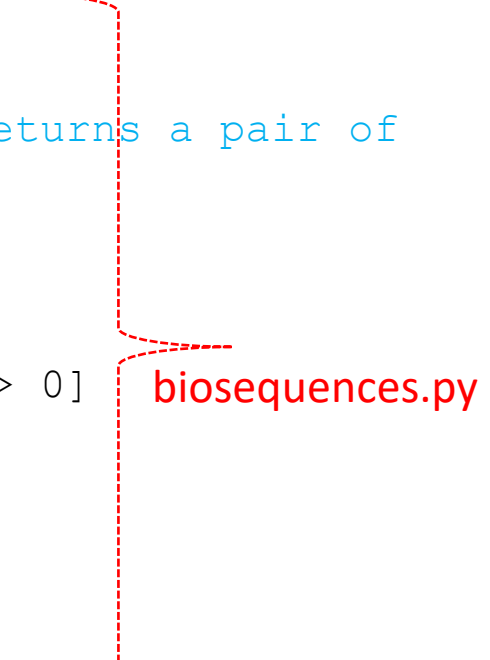
A linguagem Python permite a organização dos programas em **módulos**.

Módulos são ficheiros **.py** (ou ficheiros pré-compilados) que contêm coleções de funções úteis e relacionadas entre si, podendo também conter alguns outros objetos que não são funções, mas que contêm informação de suporte.

Permite isolamento de código e reutilização.

Módulo e **import**

```
def readFASTA(filename):  
    """  
    This function reads a FASTA format file and returns a pair of  
    strings with the header and the sequence  
    """  
    with open(filename) as a:  
        lines = [line.strip() for line in a]  
        lines = [line for line in lines if len(line) > 0]  
  
    if lines[0].startswith('>'):  
        return lines[0], ''.join(lines[1:])  
    else:  
        return '', ''.join(lines)
```



biosequences.py

Módulo e **import**

```
import biosequences
h, s = biosequences.readFASTA("gre3.txt")
print('Header:\n{}\n\nSequence:\n{}'.format(h, s))
```

imports

```
#importa o módulo datetime e adiciona-o ao namespace atual
import datetime
datetime.today.today()
datetime.timedelta(days=1)
```

```
#importa o módulo datetime e adiciona as funções date e o
timedelta ao namespace atual
from datetime import date, timedelta
date.today()
timedelta(days=1)
```

imports

```
#renomear imports  
from datetime import date  
from meu_modulo import date as minha_data
```

ERROR HANDLING

A solid orange horizontal bar at the bottom of the slide.

Error-handling

```
import datetime
import random

dia = random.choice(['Eleventh',11])
        #random entre os 2 valores fornecidos
try:
    data = 'September' + dia
    #Erro se concatenar string com int
except TypeError as err:
    data= datetime.date(2017, 9 , dia)
else:
    data+ = ' 2017'
finally:
    print (data)
```

CLASSES



Classe: Declaração

- Não tem conceito de interfaces
- Conjunto de classes especiais que começam e acabam com duplo *underscore*:
 - `__init__`, `__doc__`, `__cmp__`, `__str__`
- Atributos privados começam com duplo *underscore* mas não terminam com *underscore*

```
class User(object):  
    pass
```


Classe: Atributos

#Atributos na fase de declaração devem ser imutáveis

```
class User(object):  
    name = None  
    is_staff = False
```

Classe: Métodos

```
class User(object):  
    is_staff = False  
  
    def __init__(self, name="Anonymous"):  
        self.name = name  
        super(User, self).__init__()  
  
    def is_authorized(self):  
        return self.is_staff
```

Classe: Instanciar e aceder aos atributos

```
utilizador = User()  
  
print (utilizador.nome)  
print (utilizador.is_authorized())
```

Classe: Herança

```
class User(object):
    is_staff = False
    name = None

    def __init__(self, name="Anonymous"):
        self.name = name
        super(User, self).__init__()

    def is_authorized(self):
        return self.is_staff

class SuperUser(User):
    is_staff = True

user = User()
print (user.name , user.is_authorized())

j = SuperUser("Joao Paulo")
print (j.name, j.is_authorized())
```

REGEX

(Expressões Regulares)



Expressões Regulares

As Expressões Regulares são padrões de procura definidos para caracteres e cadeias de caracteres (strings)

As expressões regulares constituem um meio poderoso de procura e substituição de expressões complexas em grandes volumes de dados

Normalmente a expressão regular fica localizada entre duas barras / ... /

No python vamos tirar partido do módulo **re** e das suas funções

Literais (Pesquisas através de expressões regulares)

```
import re

pattern = re.compile(r"world") #compile – transforma regex em bytecode
print (pattern.search("hello world"))
```

```
import re

pattern = re.compile(r"WORLD", re.I) #re.IGNORECASE (2.x) ou re.I (3.x)
print (pattern.search("hello world"))
```

Expressões Regulares (padrões)

Expressões Regulares para caracteres isolados

- | | | |
|------|----------------|---|
| (1) | /a/ | # encontra 'a' |
| (2) | /[ab]/ | # encontra 'a' ou 'b' ([<conjunto de caracteres>]) |
| (3) | /[A-Z]/ | # encontra todas as letras maiúsculas |
| (4) | /[0-9]/ | # encontra números |
| (5) | /\d/ | # encontra números - como em 4 |
| (6) | /\D/ | # encontra tudo exceto números |
| (7) | /[0-9]\-/ | # encontra números ou o sinal de menos |
| (8) | /[\(\)]/ | # encontra tudo que estiver contido em parênteses [] |
| (9) | /[a-zA-Z0-9_]/ | # encontra letras, números ou sinal de sublinhado |

Expressões Regulares (padrões)

Expressões Regulares para caracteres isolados

(10)	/[\w]/	# encontra letras, números ou sinal de sublinhado - como em (9)
(11)	/[\W]/	# encontra tudo, exceto letras, números e sinal de sublinhado
(12)	/[\r]/	# encontra o sinal de retorno (típico do DOS)
(13)	/[\n]/	# encontra o sinal para quebra de linha
(14)	/[\t]/	# encontra o sinal de tabulação (tab)
(15)	/[\f]/	# encontra o sinal para quebra de página
(16)	/[\s]/	# encontra o sinal de espaço assim como os sinais referidos de (12) a (15)

Expressões Regulares (padrões)

Expressões Regulares para caracteres isolados

(17)	/[\\S]/	# encontra tudo, exceto sinal de espaço e os de (12) a (15)
(18)	/[äöüÄÖÜ]/	# encontra todos os caracteres com umlaut
(19)	/[^a-zA-Z]/	# encontra tudo que não contiver letras
(20)	/[ab]/s	# encontra 'a' ou 'b' também em várias linhas

Expressões Regulares (padrões)

Expressões Regulares para cadeia de caracteres

- (1) /asa/ # encontra 'asa' - também 'casa' ou 'casamento'
- (2) /asa?/ # encontra 'asa', 'casa', 'casamento' e também 'as' e 'asilo' (?)
- (3) /a./ # encontra 'as' e 'ar' (. é qualquer caractere exceto o \n)
- (4) /a+/ # encontra 'a' e 'aa' e 'aaaaa' (+ 1 ou mais ocorrências)
- (5) /a*/ # encontra 'a' e 'aa' e 'aaaaa' e 'b' (* zero ou mais ocorrências)
- (6) /ca.a/ # encontra 'casa' e 'cada', mas não 'cansa'
- (7) /ca.+a/ # encontra 'casa', 'cada', 'cansa' e 'canela'
- (8) /ca.?o/ # encontra 'caso', 'cabo', 'cao' e 'cano' mas não 'canelado'

Expressões Regulares (padrões)

Expressões Regulares para cadeia de caracteres

- (9) `/x{10,20}/` # encontra sequências de 10 a 20 'x'
- (10) `/x{10,}/` # encontra sequências de 10 ou mais 'x'
- (11) `/x{2}y/` # encontra 'xxxy', 'axxy', 'sadsaxxy', 'dfdaaxxydfa'
- (12) `/Clara\b/` # encontra 'Clara' mas não 'Clarazinha'
- (13) `/\bassa/` # encontra 'assa' ou 'assado' mas não 'massa'
- (14) `/\bassa\b/` # encontra 'assa' mas não 'assado' e nem 'massa'
- (15) `/\bassa\B/` # encontra 'assado' mas não 'assa' e nem 'massa'
- (16) `/^Julia/` # encontra 'Julia' apenas no início do contexto da pesquisa

Expressões Regulares (padrões)

Expressões Regulares para cadeia de caracteres

(17) `/Helena$/` # encontra 'Helena' apenas no final do contexto da pesquisa

(18) `/^\s*$ /` # encontra linhas constituídas apenas por zero ou mais espaços

Expressões Regulares (padrões)

Regex com alternativas

`/a|b/` # encontra 'a' ou 'b' - idêntico a `/[ab]/`

`/com|sem/` # encontra 'com' e 'descompensar', como também 'sem' e 'semântica'

Precedências e uso de parenteses

`/a|bc|d/` # encontra 'a' ou 'bc' ou 'd'

`/(a|b)(c|d)/` # encontra 'ac' ou 'ad' ou 'bc' ou 'bd'

Expressões Regulares (padrões)

Meta caracteres

<code>^</code>	Início da linha
<code>\$</code>	Fim da linha
<code>\b</code>	Fronteiras de palavras (espaço como delimitador)
<code>\B</code>	Contrário do <code>\b</code>
<code>\A</code>	Início do input
<code>\Z</code>	Fim do input

Expressões Regulares

Flags compilação da regex (re.compile)

re.IGNORECASE ou re.I	Pesquisa independentemente da escrita maiúsculas ou minúsculas
re.MULTILINE ou re.M	Assume os caracteres ^ (inicio) e \$ (fim) para múltiplas linhas
re.DOTALL ou re.S	Altera o comportamento do . na regex (assume também o \n)
re.LOCALE ou re.L	O \w, \W, \b, \B, \s, e \S ficam dependentes dos <i>settings</i> locais (língua)
re.VERBOSE ou re.X	Ignora eventuais espaços e # que possam existir na regex
re.DEBUG	Fornece informação sobre o padrão de compilação
re.UNICODE ou re.U	O comportamento do \w, \W, \b, \B, \s, e \S rege-se pelo unicode
re.ASCII ou re.A	O comportamento do \w, \W, \b, \B, \s, e \S rege-se pelo ASCII

Expressões Regulares – Funções

search

```
import re

pattern = re.compile(r"world")
print (pattern.search("hello world"))
```

match

```
import re

pattern = re.compile(r"\bworld")
print (pattern.match("hello world"))
```

Expressões Regulares – Funções

findall

```
import re

pattern = re.compile(r'hello')
print (pattern.findall("hello let me say hello to you again"))
```

split

```
import re

pattern = re.compile(r"\W")
print (pattern.split("hello world"))
```

Expressões Regulares – Funções

finditer

```
import re

pattern = re.compile(r"(\w+) (\w+)")
it = pattern.finditer("Hello world hola mundo")

match = it.next()
print (match.groups())
print (match.span())

match = it.next()
print (match.groups())
print (match.span())
```

Expressões Regulares – Funções

sub (find and replace)

```
import re

pattern = re.compile(r"[0-9]+")
print (pattern.sub("-", "order0 order1 order13"))
```

```
import re

pattern = re.compile(r"^")
print (pattern.sub("00351", "961231321"))
```

Expressões Regulares – Backtrack

Backtrack:

Consiste em capturar grupos dentro da *regex* que podem ser usados mais tarde sob a forma de variáveis

Podem ser adotados nomes de variáveis para cada grupo

Expressões Regulares – Backtrack

```
import re

str = "8120123;Ana Rita"
pattern = re.compile(r'^(.*);(.*)$') #parenteses permite criar grupos
print pattern.sub(r"\2;\1",str)      #1º parenteses é o \1 e o segundo o \2
```

```
import re

str = "8120123;Ana Rita"
pattern = re.compile(r'^(?P<numero>.*);(?P<nome>.*)$') #utilizados nomes de variáveis
print pattern.sub(r"\g<nome>;\g<numero>",str)
```

Expressões Regulares – Backtrack

```
import re

str = "8120123;Ana Rita"

pattern = re.compile(r'^(?P<numero>.*);(?P<nome>.*)$')
lista_grupos = pattern.match(str)

if lista_grupos:
    print (lista_grupos.group('nome'), lista_grupos.group(1))
```

Referências

<https://www.slideshare.net/nowells/introduction-to-python-5182313>

<http://webpages.fc.ul.pt/~aeferreira/python/index.html>